# ParaText: Scalable Text Modeling and Analysis

Daniel M. Dunlavy
Comp. Sci. & Informatics
Sandia National Laboratories
P.O. Box 5800, M/S 1318
Albuquerque, NM 87185-1318
dmdunla@sandia.gov

Timothy M. Shead
Data Analysis & Visualization
Sandia National Laboratories
P.O. Box 5800, M/S 1323
Albuquerque, NM 87185-1323
tshead@sandia.gov

Eric T. Stanton
Data Analysis & Visualization
Sandia National Laboratories
P.O. Box 5800, M/S 1323
Albuquerque, NM 87185-1323
etstant@sandia.gov

## ABSTRACT

Automated analysis of unstructured text documents (e.g., web pages, newswire articles, research publications, business reports) is a key capability for solving important problems in areas including decision making, risk assessment, social network analysis, intelligence analysis, scholarly research and others. However, as data sizes continue to grow in these areas, scalable processing, modeling, and semantic analysis of text collections becomes essential. In this paper, we present the ParaText text analysis engine, a distributed memory software framework for processing, modeling, and analyzing collections of unstructured text documents. Results on several document collections using hundreds of processors are presented to illustrate the flexibility, extensibility, and scalability of the the entire process of text modeling from raw data ingestion to application analysis.

## Categories and Subject Descriptors

I.2.7 [**Computing Methodologies**]: Natural Language Processing—*text analysis*

## General Terms

Algorithms, Design, Performance, Text Analysis

## 1. INTRODUCTION

Automated processing, modeling, and analysis of unstructured text (news documents, web content, journal articles, etc.) is a key task in many data analysis and decision making applications. In many cases, documents are modeled as term or feature vectors and latent semantic analysis (LSA) [4] is used to model latent, or hidden, relationships between documents and terms appearing in those documents. LSA supplies conceptual organization and analysis of document collections by modeling high-dimension feature vectors in many fewer dimensions.

In this paper, we emphasize scalability of the full LSA process from ingesting text and modeling data, to analysis tasks including information retrieval and document similarity. We have implemented several alternate methods for parts of the process that require significant inter-processor communication, and present strong scaling studies for these methods.

## 2. RELATED WORK

Past work on the scalability of LSA modeling has focused on the computation of a truncated singular value decomposition (SVD) of the term-by-document matrix modeling the data. (e.g., [2, 1, 11]). Other approaches for increasing the computational performance of LSA include alternatives to the SVD for dimensionality reduction (e.g., [12]) and feature selection to reduce the size of the term-document matrices (e.g., [14]).

Work on fully parallel text analytic systems is less common. Berry and Martin developed the Parallel General Text Processor (PGTP) for LSA modeling [2], but the focus of that work was on parallelization of the SVD within a text analysis system. Krishnan *et al.* have developed parallel text analysis capabilities in the IN-CITE visual analytics software application [7], where global arrays and distributed hashmaps are used to improve overall performance of the system. Their work was demonstrated on up to 32 processors. The UIMA Grid project [5] is an alternate approach that leverages grid computing to enable asynchronous parallel processing of text for entity extraction; results were presented on only a few processors.

The goal of our work is to investigate the use of distributed memory architectures for text analysis and modeling. We describe the components of the ParaText system and demonstrate its performance through several strong scaling studies.

## 3. THE PARATEXT SYSTEM

The ParaText system is comprised of a collection of text analysis components designed to function within a Titan data processing pipeline [13], where data sources, filters, and sinks can be combined in arbitrary ways . The ParaText components can be used either as a C++ programming library, or via a web service that implements a RESTful API [6] atop an Apache httpd server. Thus, the ParaText capabilities outlined in this report can be accessed using a variety of programming languages and environments.

Throughout this paper, we denote $n$ as the number of documents and $m$ as the number of unique terms, and $p$ as the number of processors used in the computations.

## 3.1 Text Extraction

The first part of the pipeline consists of filters for extracting and transforming text. With the exception of determining which files should be processed on which processors, the filters described in this section all parallelize extremely well.

*Document Ingestion.* The DOCUMENT INGESTION filter is responsible for partitioning a set of documents and loading them into memory as a table where each row corresponds to a document. We have implemented several partitioning strategies that control how processors determine which files to load locally. The *Documents* partitioning strategy does a simple round-robin distribution where each process loads $1/p$ documents from the set. This strategy is simple to implement and requires no communication, but can lead to imbalanced loading as some processors may accumulate documents that are smaller- or larger-than-average. The *Bytes* partitioning strategy tries to balance loading by assigning files to processors so that each processor receives roughly the same number of input bytes. Because this is a variation on bin packing — a combinatorial NP hard problem — we use a heuristic approach of maintaining a "bucket" for each processor, then inserting each file, in descending order of file size, into whichever bucket contains the fewest number of file bytes at the time. Early versions of this approach (which we call *Thrash*) did not require communication, but performed poorly due to filesystem contention as every processor simultaneously tried to retrieve the size of every file in the set. Subsequent versions use a single processor to retrieve file sizes and distribute them to the remaining processes before beginning the bucketing process.

*Text Extraction.* Once the local table of documents to be loaded has been created, we use MIME type information to extract text, using the TEXT EXTRACTION filter. This filter contains a collection of strategy objects, each of which is responsible for extracting text from documents of a given MIME type. Note that the text extraction strategies can perform arbitrarily-complex operations to extract text from a document, including extracting text from binary file formats such as PDF or word-processing documents, extracting metadata from images, or even performing optical character recognition on the contents of an image. For the experiments presented here, we relied on a default extraction strategy that handles all `text/*` MIME types. The extracted text is stored as Unicode [10] strings using UTF-8 encoding, so the system is capable of working with mixed-language text.

*Tokenization.* Following text extraction, the TOKENIZATION filter converts document text into a table of tokens. Tokenization is performed by splitting the document text into tokens using delimiters specified as half-open ranges of Unicode codepoints. Delimiters can be kept, allowing for tokenization of logosyllabic scripts such as Chinese, Korean, and Japanese by specifying ranges of logograms as "kept" delimiters, so that individual glyphs become tokens.

*Token Length Filtering.* We use two instances of the TOKEN LENGTH filter to discard tokens that are either too short or too long. This improves the downstream analysis by reducing noise in the data models.

*N-Gram Extraction.* The N-GRAM EXTRACTION filter converts individual tokens into $n$-grams and is parameterized to allow arbitrary values for $n$. We used unigrams ($n = 1$) for all experiments in this paper.

*Case Folding.* We use the CASE FOLDING filter to transform the resulting tokens to a form where they can be used in case-insensitive comparisons. This transformation is carried-out using the rules provided by Unicode, so the results can be used for case-insensitive comparisons across all Unicode-supported languages.

*Token Value Filtering.* To provide filtering of stop-words, we use the TOKEN VALUE filter, which is parameterized by a list of tokens to be discarded. We used the standard stop word list from the SMART project [9].

## 3.2 Term Dictionary Creation

Once each processor determines the list of terms in its local data (i.e., documents), the TERM DICTIONARY filter creates a global dictionary where each term is listed exactly once. Because this process necessitates communication of large numbers of strings between processors, we created several different implementations for testing: in *N-to-1*, every processor sends its local terms to processor 0, which creates the global dictionary and broadcasts the results back to every processor. For *N-to-N*, each processor broadcasts its local terms to all other processors, which then create their own copies of the global dictionary. In the *Binary Tree* approach, each processor sends its local terms to a "neighbor", which consolidates them with its own local terms, sending the results to a "super neighbor", and-so-on until the complete global dictionary has been created on one process that broadcasts the results to the others. The *Round Robin* approach involves processor $k$ sending its local terms to processor $(k+1) \mod p$, where they are consolidated with the local terms. This process runs $p$ times, so that every term eventually reaches every processor. Finally, we have a *MapReduce* approach that uses the MapReduce-MPI library [8] to consolidate and distribute terms.

## 3.3 Term Document Matrix Creation

Given the list of local terms and the global term dictionary computed in the TERM DICTIONARY filter, each processor uses the TERM DOCUMENT MATRIX filter to create its portion of a sparse, distributed term-document frequency matrix (no inter-processor communication is required). For each term from the local term list, the global term dictionary is used to determine the corresponding matrix row. Two methods are implemented for term dictionary lookup: *Global* lookup is a naive approach where the global term dictionary is used to lookup each term with $O(m \log m)$ performance; *Global+Local* lookup is a more sophisticated two-stage approach where local lookup results are cached in a smaller lookup table to improve performance.

## 3.4 Term Weighting

Once the term-document frequency matrix is generated, it must be weighted to incorporate the importance of the terms throughout the collection. In this paper, we focus on the standard *log-entropy* weighting scheme [4] employed in many LSA studies, which illustrates the challenges associated with term weighting on distributed memory architectures. This

weighting scheme involves the product of local quantities (frequencies of terms within each document) and global quantities (entropies of terms across the entire document collection). In ParaText, the local and global computations are separated into different filters: the LOG WEIGHTING and ENTROPY WEIGHTING filters, respectively.

The entropy of term $i$ across the collection is defined as

$$g_i = -\frac{1}{n}\sum_{j=1}^{n}\frac{tf_{ij}}{gf_i}\log\frac{tf_{ij}}{gf_i}\ , \qquad (1)$$

where $tf_{ij}$ is the frequency of term $i$ in document $j$ and $gf_i$ is the global frequency of term $i$ across the collection. Interprocessor communication is required both in computing $gf_i$ for each term and the sum in $g_i$ for each term. We have implemented several methods to study the impact of these communication requirements. In the *N-to-1* method, every processor computes its local values of $gf_i$ and sends those to processor 0, which sums the values and broadcasts the results back to every processor. The sums for $g_i$ are then computed in a similar fashion. In the *N-to-N* method, $gf_i$ and $g_i$ are first computed locally and then results are broadcast to all other processors for computing the global values. In both methods, there is the option to broadcast the locally computed values using either dense or sparse vectors. Once the local and global term weights are computed, the SCALE DIMENSION filter then applies these weights to the matrix.

## 3.5 Singular Value Decomposition

To compute the SVD of the weighted term-document matrix, $A$, ParaText wraps the distributed block Krylov Schur method from the Anasazi package of the Trilinos solver library [1]. Using shallow copies of data into the sparse matrix class in Trilinos, we avoid data replication. The rank-$k$ truncated SVD of $A$ is computed as $A_k \approx U_k\Sigma_k V_k^{\mathsf{T}}$, where $U_k$, $\Sigma_k$, and $V_k$ are matrices containing the left singular vectors, singular values, and right singular vectors, respectively.

## 3.6 Corpus Analysis

*Document Similarities.* An important application of text modeling in general and LSA in particular [3] is determination of the topical or conceptual relationships between documents in a large collection. To model these relationships, pairwise document similarity or distance measures are often computed. In ParaText, document similarities are computed as the cosine values between scaled LSA document vectors (in $V_k\Sigma_k$). Thus, the similarity between documents $i$ and $j$ are computed as $\langle\hat{V}_i,\hat{V}_j\rangle/(\|\hat{V}_i\|\ \|\hat{V}_j\|)$, where $\langle\cdot,\cdot\rangle$ is the standard inner product and $\hat{V}_i$ is the $i$th row of $V_k\Sigma_k$.

# 4. RESULTS

*Computing Environment.* The system we used for testing is comprised of 256 compute nodes, each with a Dual 3.6 GHz Intel EM64T processor and 6 GB RAM. The system's high-speed message passing fabric is Infiniband, and the file system is Lustre with a bandwidth of 15 GB/second.

*Data.* The data used in the experiments presented here consist of a subset of HTML documents in the 2007 Spock Challenge test set (http://challenge.spock.com/). For experiments involving 64 processors, $n = 2458$ and $m = 669940$ (0.12% matrix density); for those on 512 processors, $n = 45945$ and $m = 4440327$ (0.017% matrix density) were used. Note that this decrease in matrix density for more documents is typical in text analysis.

*Strong Scaling Studies.* Figure 1 presents the results of strong scaling studies using 64 processors for the filters with multiple implementations. The plots all show mean CPU times (with error bars denoting standard error) over 3 runs as a function of the number of processors.

There are significant differences in the document partitioning methods (Fig. 1a), where partitioning by *Documents* appears the most scalable. Also the *N-to-N* methods appear

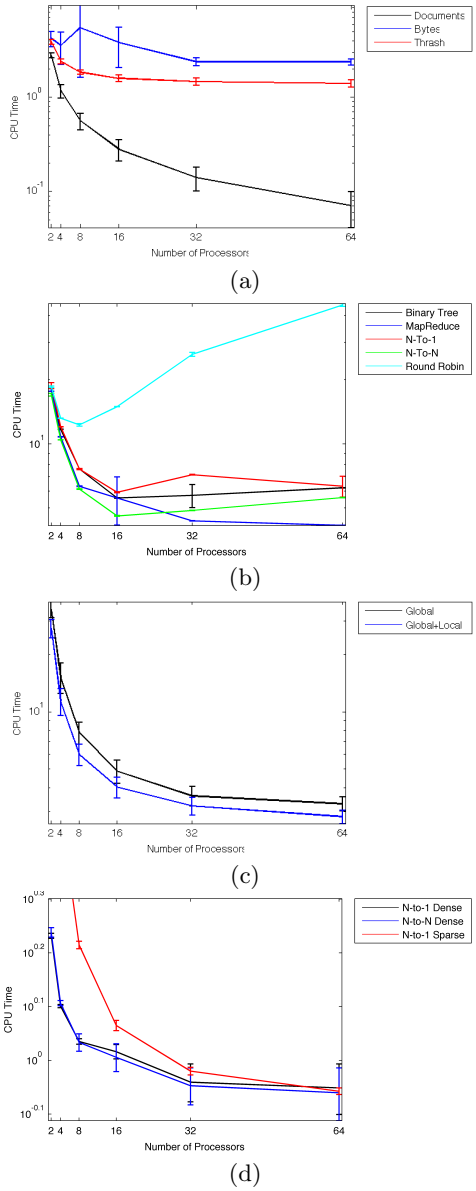


**Figure 1: Strong scaling studies for methods associated with (a) document partitioning, (b) term dictionary creation, (c) frequency matrix creation and (d) entropy weighting.**

to perform slightly (but not statistically significantly) better than the *N-to-1* methods (Figs. 1b and 1d), even though the former methods require more overall communication. In terms of sizes of packets being communicated, using dense over sparse arrays in the ENTROPY WEIGHTING filter appears better for fewer processors (Fig. 1d). However, as the number of processors increases (and thus the local term dictionaries become more sparse due to fewer documents on each processor), the sparse data passing has potential for improved performance (as demonstrated by the trajectory of improvement in the figure). Since *MapReduce* for term dictionary creation seems promising as the number of processors increases (Fig. 1b), we plan to explore additional use of MapReduce in future versions of ParaText where appropriate. Finally, caching of local information reduces the overall costs associated with distributed term dictionary creation (Fig. 1c), and we will be investigating more use of this throughout the ParaText pipeline.

Figure 2 presents the results of strong scaling studies for all pipeline filters. Figure 2a illustrates that for the larger problem using 512 processors, most of the filters requiring little or no inter-processor communication achieve strong scalability as expected, although improvement is still possible for the document ingestion and term document matrix creation. For the filters requiring significant inter-processor communication, we see that more work is needed to achieve useful speedups as we increase the number of processors (Figure 2b). We leave this as future work.

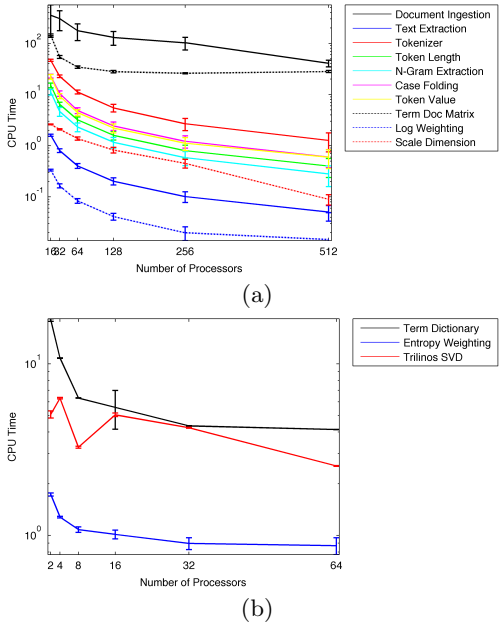


**Figure 2: Strong scaling studies for the ParaText pipeline illustrating performance of filters with (a) little or no inter-processor communication and (b) significant inter-processor communication.**

## 5. CONCLUSIONS

We have presented the ParaText system, an end-to-end process for scalable distributed memory analysis of large document collections. Through strong scaling studies, we have illustrated that there are significant challenges associated with LSA scalability and identified several areas for improvement in document ingestion, term-dictionary creation (e.g., using distributed merge sort), and matrix computations (e.g., tuned matrix vector products).

## 6. ACKNOWLEDGMENTS

This work was funded by the Laboratory Directed Research & Development (LDRD) program at Sandia National Laboratories, a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

## 7. REFERENCES

[1] C. G. Baker, U. L. Hetmaniuk, R. B. Lehoucq, and H. K. Thornquist. Anasazi software for the numerical solution of large-scale eigenvalue problems. *ACM TOMS*, 36(3):13:1–13:23, 2009.

[2] M. W. Berry and D. I. Martin. Parallel SVD for scalable information retrieval. In *Proc. Intl. Workshop on Parallel Matrix Algorithms and Applications*, Neuchatel, Switzerland, 2000.

[3] P. Crossno, D. Dunlavy, and T. Shead. LSAView: A tool for visual exploration of latent semantic modeling. In *Proc. IEEE VAST*, 2009.

[4] S. T. Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers*, 23(2):229–236, 1991.

[5] M. T. Egner, M. Lorch, and E. Biddle. Uima grid: Distributed large-scale text analysis. In *Proc. of the 7th IEEE International Symposium on Cluster Computing and the Grid*, pages 317–326, Washington, DC, USA, 2007. IEEE Computer Society.

[6] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM TOIT*, 2(2):115–150, 2002.

[7] M. Krishnan, S. Bohn, W. Cowley, and J. Crow, V.and Nieplocha. Scalable visual analytics of massive textual datasets. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–10, 26-30 March 2007.

[8] S. Plimpton and K. Devine. MapReduce-MPI Library. http://www.sandia.gov/~sjplimp/mapreduce.html.

[9] G. Salton, editor. *The SMART Retrieval System: Experiments in Automatic Document Processing.* Prentice-Hall, 1971.

[10] The Unicode Consortium. *The Unicode Standard, Version 5.0 (5th Edition).* Addison-Wesley Professional, 2006.

[11] S. Vigna. Distributed, large-scale latent semantic analysis by index interpolation. In *Proc. InfoScale*, pages 1–10, 2008.

[12] D. Widdows and K. Ferraro. Semantic vectors: a scalable open source package and online technology management application. In *Proc. LREC*, 2008.

[13] B. Wylie and J. Baumes. A unified toolkit for information and scientific visualization. In *SPIE*, 2009.

[14] J. Yan, S. Yan, N. Liu, and Z. Chen. Straightforward feature selection for scalable latent semantic indexing. In *Proc. SDM*, pages 1159–1170, 2009.